MDF: Magnetic Particle Imaging Data Format

T. Knopp[1,2], T. Viereck[3], G. Bringout[4], M. Ahlborg[5], A. von Gladiss[5], C. Kaethner[5], A. Neumann[5], P. Vogel[6], J. Rahmer[7], M. Möddel[1,2]

[1]Section for Biomedical Imaging, University Medical Center Hamburg-Eppendorf, Germany
[2]Institute for Biomedical Imaging, Hamburg University of Technology, Germany
[3]Institute of Electrical Measurement and Fundamental Electrical Engineering, TU Braunschweig, Germany
[4]Physikalisch-Technische Bundesanstalt, Berlin, Germany
[5]Institute of Medical Engineering, University of Lübeck, Germany
[6]Department of Experimental Physics 5 (Biophysics), University of Würzburg, Germany
[7]Philips GmbH Innovative Technologies, Research Laboratories, Hamburg, Germany

May 28, 2021

Version **2.1.0**

Abstract

Magnetic particle imaging (MPI) is a tomographic method for determining the spatio-temporal distribution of magnetic nanoparticles. In this document, a file format for the standardized storage of MPI and magnetic particle spectroscopy (MPS) data is introduced. The aim of the Magnetic Particle Imaging Data Format (MDF) is to provide a coherent way of exchanging MPI and MPS data acquired with different devices worldwide. The focus of the MDF is on sequence parameters, measurement data, calibration data, and reconstruction data. The format is based on the hierarchical document format in version 5 (HDF5). This document discusses the MDF version 2.1.0, which is not backward compatible with version 1.x.y.

1 Introduction

The purpose of this document is to introduce a file format for exchanging Magnetic Particle Imaging (MPI) and Magnetic Particle Spectroscopy (MPS) data. The Magnetic Particle Imaging Data Format (MDF) is based on the hierarchical document format (HDF) in version 5 [1]. HDF5 is able to store multiple datasets within a single file providing a powerful and flexible data container. To allow an easy exchange of MPI data, one has to specify a naming scheme within HDF5 files which is the purpose of this document. In order to create and access HDF5 data, an Open Source C library is available that provides dynamic access from most programming languages. MATLAB supports HDF5 by its functions `h5read` and `h5write`. For Python, the `h5py` package exists, and the Julia programming language provides access to HDF5 files via the `HDF5` package. For languages based on the .NET framework, the `HDF5DotNet` library is available.

The MDF is mainly focused on storing measurement data, calibration data, or reconstruction data together with the corresponding sequence parameters and metadata. Even though it is possible to combine measurement data and recon-

struction data into a single file, it is recommended to use a single file for each of the following dataset types:

1. Measurement data
2. Calibration data
3. Reconstruction data

1.1 Datatypes & Storage Order

MPI parameters are stored as regular *HDF5 datasets*. *HDF5 attributes* are not used in the current specification of the MDF. For most datasets, a fixed datatype is used, for example the drive-field amplitudes are stored as `H5T_NATIVE_DOUBLE` values. Whenever a data type has a big- and little-endian version, the little-endian data type should be used. For our convenience, we refer to the HDF5 datatypes `H5T_STRING`, `H5T_NATIVE_DOUBLE`, and `H5T_NATIVE_INT64` as `String`, `Float64` and `Int64`. Boolean data is stored as `H5T_NATIVE_INT8`, which we refer to as `Int8`.

The datatype of the MPI measurement and calibration data offers more freedom and is denoted by `Number`, which can be any of the following HDF5 data types: `H5T_NATIVE_FLOAT`, `H5T_NATIVE_DOUBLE`, `H5T_NATIVE_INT8`, `H5T_NATIVE_INT16`, `H5T_NATIVE_INT32`, `H5T_NATIVE_INT64` or a complex number as defined below. In the case that the calibration data is compressed, the indices are stored as `Integer`, which can be any of the following HDF5 data types: `H5T_NATIVE_INT8`, `H5T_NATIVE_INT16`, `H5T_NATIVE_INT32`, `H5T_NATIVE_INT64`.

Since storing complex data in HDF5 is not standardized, we define a compound datatype `H5T_COMPOUND` in HDF5 with fields `r` and `i` using one of the above mentioned number types to represent the real and the imaginary part of a complex number. This representation was chosen because it is also the default behavior for complex numbers in Python using numpy and h5py as well as in Julia using the HDF5.jl package. `Complex128` refers to the complex compound type with base type `H5T_NATIVE_DOUBLE`.

For later identification of a data set, we store three Universally Unique Identifiers (UUIDs) (RFC 4122) [2] in its canonical textual representation as 32 hexadecimal digits, displayed in five groups separated by hyphens `8-4-4-4-12` as for example `ee94cb6d-febf-47d9-bec9-e3afa59bfaf8`. For the generation of the UUIDs, we recommend to use version 4 of the UUID specification.

Whenever multidimensional data is stored, dimensions are arranged in a way that cache is utilized best for fast reconstruction or fast frame selection for example. The leading dimension in the MDF specification is slowest to access and the last dimension is the fastest to access (contiguous memory access). That also means that dependent on the memory layout of your programming language the order of the dimensions will be just as in the MDF specifications (row major) or reversed (column major). Please take this into account when reading or writing MDF files, which includes the usage of HDF5 viewers.

1.2 Units

With one exception, physical quantities are given in SI units. The magnetic field strength is reported in $\mathrm{T}\mu_0^{-1} = 4\pi\mathrm{Am}^{-1}\mu_0^{-1}$. This convention has been proposed in the first MPI publication [3] and consistently been used in most MPI related publications. The aim of this convention is to report the numbers on a Tesla scale, which most readers with a background in magnetic resonance imaging are familiar with.

1.3 Optional Parameters

The MDF has 8 main groups in the root directory and 2 sub-groups. We distinguish between optional and non-optional groups as well as optional, non-optional, and conditional parameters.

Any optional parameter can be omitted, whereas any non-optional parameter in a non-optional group is mandatory. Conditional parameters are linked to Boolean parameters and have to be provided, if these parameters are true and can be omitted if they are false. If a parameter is optional, non-optional, or conditional is indicated by yes, no, or the corresponding Boolean parameter.

If a group is optional all of its parameters may be omitted, if this group is not used. The groups `/`, `/study`, `/experiment`, `/scanner`, `/acquisition` contain mostly metadata and are mandatory. The `/tracer` group is only mandatory if magnetic material has been placed in the MPI system. The groups `/measurement`, `/calibration`, and `/reconstruction` are all optional. In case of calibration measurements, the `/calibration` group is mandatory. The reconstruction data is stored in `/reconstruction`.

1.4 User defined parameters

It is possible to store user defined parameters in an MDF. For instance one may want to store the temperature of the room in which your MPI device is operated. In this case, one is free to add new parameters to any of the existing groups. Moreover, if necessary, one can also introduce new groups. In order to be able to distinguish these datasets and groups from the specified ones, it is mandatory to use the prefix "_" for all parameters and groups. As an example, one could add a new group `/_room` that includes the dataset `_temperature`. Using the prefix "_" will ensure that the stored dataset is compatible with future versions of the MDF.

1.5 Naming Convention

Several parameters within the MDF are linked in dimensionality. We use short variable names to indicate these connections. The following table describes the meaning of each variable name used in this specification.

Variable	Number of
A	tracer materials/injections for multi-color MPI
N	acquired frames ($N = O + E$), same as a spatial position for calibration
E	acquired background frames ($E = N - O$)
O	acquired foreground frames ($O = N - E$)
B	coefficients stored after sparsity transformation ($B \leq O$)
J	periods within one frame
Y	partitions of each patch position
C	receive channels
D	drive-field channels
F	frequencies describing the drive-field waveform
V	points sampled at receiver during one drive-field period
W	sampling points containing processed data ($W = V$ if no frequency selection or bandwidth reduction has been applied)
K	frequencies describing the processed data ($K = V/2 + 1$ if no frequency selection or bandwidth reduction has been applied)
Q	frames in the reconstructed MPI data set
P	voxels in the reconstructed MPI data set
S	channels in the reconstructed MPI data set

1.6 Contact

If you find mistakes in this document or the specified file format or if you want to discuss extensions or improvements to this specification, please open an issue on GitHub:

`https://github.com/MagneticParticleImaging/MDF`

As the file format is versionized it will be possible to extend it for future needs of MPI. The current version discussed in this document is version 2.1.0.

1.7 arXiv

As of version 1.0.1, the most recent release of these specifications can also be also found at:

`https://arxiv.org/abs/1602.06072`

If you use MDF, please cite this document using the arXiv reference, which is also available for download as `MDF.bib` from GitHub.

1.8 Code examples

If you want to get a basic impression of how to handle MDF files you can visit the gitub repository of the MDF project:

`https://github.com/MagneticParticleImaging/MDF`

There you will find the example directory, which contains a code example written in Julia, MATLAB, and Python. More details can be found in the `README` of the repository.

1.9 Reference Implementation

A reference implementation for a high level MDF access is available at:

`https://github.com/MagneticParticleImaging/MPIFiles.jl`

`MPIFiles.jl` [4] is a package written in the programming language Julia [5, 6, 7]. It can read MDF V1, MDF V2, and the dataformat of Bruker MPI systems using a common interface. It also provides functions to convert MDF V1 or Bruker MPI files into MDF V2.

2 Data (group: /)

Remarks: Within the root group, the metadata about the file itself is stored. Within several subgroups, the metadata about the experimental setting, the MPI tracer, and the MPI scanner can be provided. The actual data is stored in dedicated groups about measurement data and reconstruction data.

Parameter	Type	Dim	Unit/Format	Optional	Description
`time`	String	1	yyyy-mm-ddThh:mm:ss.ms	no	UTC creation time of MDF data set
`uuid`	String	1	3170fdf8-f8e1-4cbf-ac73-41520b41f6ee	no	Universally Unique Identifier (RFC 4122) of MDF file
`version`	String	1	2.1.0	no	Version of the file format

2.1 Study Description (group: `/study/`, non-optional)

Remarks: A study is supposed to group a series of experiments to support, refute, or validate a hypothesis. The study group should contain `name`, `number`, `uuid`, and `description` of the study.

Parameter	Type	Dim	Unit/Format	Optional	Description
`description`	String	1		no	Short description of the study
`name`	String	1		no	Name of the study
`number`	Int64	1		no	Number of the study
`time`	String	1	yyyy-mm-ddThh:mm:ss.ms	yes	UTC creation time of study
`uuid`	String	1	295258fe-b650-4e5f-96db-b83f11089a6c	no	Universally Unique Identifier (RFC 4122) of study

2.2 Experiment Description (group: `/experiment/`, non-optional)

Remarks: For each experiment within a study a `name`, `number`, `uuid`, and `description` have to be provided. Additionally, the name of the `subject` imaged and the flag `isSimulation` indicating if data has been obtained via simulations have to be stored.

Parameter	Type	Dim	Unit/Format	Optional	Description
description	String	1		no	Short description of the experiment
isSimulation	Int8	1		no	Flag indicating if the data in this file is simulated rather than measured
name	String	1		no	Experiment name
number	Int64	1		no	Experiment number within study
subject	String	1		no	Name of the subject that was imaged
uuid	String	1	f96dbc48-1ebd-44c7-b04d-1b45da054693	no	Universally Unique Identifier (RFC 4122) of experiment

2.3 Tracer Parameters (group: /tracer/, optional)

Remarks: The tracer parameter group contains information about the MPI tracers used during the experiment. For each of the A tracers `name`, `batch`, `vendor`, `volume`, and molar `concentration` of `solute` per liter must be provided. Additionally, the time point of injection can be stored.

This version of the MDF can handle two basic scenarios. In the first one, static tracer phantoms are used. In this case, the phantom contains A distinct tracers. For example, these might be particles of different core sizes, mobile or immobilized particles. In this case, `injectionTime` is not used. In the second case, A boli (e.g. pulsed boli) are administrated during the measurement, in which case the approximate administration volume, tracer type and time point of injection can be provided. Note that the injection clock recording the injection time should be synchronized with the clock, which provides the starting time of the measurement.

In case of a background measurement with no applied tracers in the scanner, the optional tracer group can be omitted.

Parameter	Type	Dim	Unit/Format	Optional	Description
batch	String	A		no	Batch of tracer
concentration	Float64	A	mol(`solute`)/L	no	Molar concentration of `solute` per litre
injectionTime	String	A	yyyy-mm-ddThh:mm:ss.ms	yes	UTC time at which tracer injection started
name	String	A		no	Name of tracer used in experiment
solute	String	A		no	Solute, e.g. Fe
vendor	String	A		no	Name of tracer supplier
volume	Float64	A	L	no	Total volume of applied tracer

2.4 Scanner Parameters (group: `/scanner/`, non-optional)

Remarks: The scanner parameter group contains information about the MPI scanner used, such as `name`, `manufacturer`, `boreSize`, field `topology`, `facility` where the scanner is installed, and the `operator`.

Parameter	Type	Dim	Unit/Format	Optional	Description
`boreSize`	`Float64`	1	m	yes	Diameter of the bore
`facility`	`String`	1		no	Facility where the MPI scanner is installed
`manufacturer`	`String`	1		no	Scanner manufacturer
`name`	`String`	1		no	Scanner name
`operator`	`String`	1		no	User who operates the MPI scanner
`topology`	`String`	1		no	Scanner topology (e.g. FFP, FFL, MPS)

2.5 Acquisition Parameters (group: `/acquisition/`, non-optional)

Remarks: The acquisition parameter group can describe different imaging protocols and trajectory settings. The corresponding data is organized into general information, a subgroup containing information on the D excitation channels, and a subgroup containing information on the C receive channels.

The term *frame* refers to the data collected during one acquisition period. Usually all data within a frame will be combined to reconstruct a single MPI image/tomogram. In the simplest scenario, this data is acquired during one `drivefield/cycle`. If block-averaging is applied the amount of data captured stays the same while the acquisition time increases by a factor of `numAverages`. In this document we will refer to the product of one `drivefield/cycle` and `numAverages` as *one drive-field period*.

In case that the static selection field is changed over time one will measure several drive-field periods each with a different gradient field setting. In a stepped multi-patch sequence the gradient field will be spatially shifted and/or scaled in between drive-field periods and remain constant within. In such a scenario a full frame consists of `numPeriodsPerFrame` (J) periods. The gradient scaling or shift within each period are described by the fields `/acquisition/gradient` and `/acquisition/offsetField`. The former has $J \times Y \times 3 \times 3$ entries whereas the later is of dimension $J \times Y \times 3$. While J labels the drive-field periods, Y can be used to describe dynamic multi-patch sequences where the gradient and offset field change during a drive-field period. Each period is discretized into Y equidistantly spaced time intervals. The values in `/acquisition/gradient` and `/acquisition/offsetField` describe the gradient and offset field at the beginning of each time interval. In case $Y = 1$ the fields at the beginning of the period J are provided, which can be used to describe a static multi-patch sequence. Higher numbers of Y allow the description of an arbitrarily fine grained gradient and offset field sequence.

It should be noted that the combined gradient and offset field can be expressed locally around the scanner center $\mathbf{r}_c \in \mathbb{R}^3$ via the Taylor expansion

$$\mathbf{H}(\mathbf{r}) = \mathbf{H}(\mathbf{r}_c) + \mathbf{J}_\mathbf{H}(\mathbf{r}_c)(\mathbf{r} - \mathbf{r}_c) + \dots, \tag{1}$$

where $\mathbf{J}_\mathbf{H}(\mathbf{r}_c)$ is the Jacobian of $\mathbf{H}$ at $\mathbf{r}_c$. `/acquisition/offsetField` stores the field $\mathbf{H}(\mathbf{r}_c)$ and `/acquisition/gradient` stores the 3×3 matrix $\mathbf{J}_\mathbf{H}$. Both parameters are for most scanner topologies sufficient to uniquely describe the applied field sequence. In case of non-linear fields and/or a higher degree of freedom in the number of applied field generators, one may need additional custom fields in order to uniquely define the sequence parameters.

Parameter	Type	Dim	Unit/Format	Optional	Description
`gradient`	`Float64`	$J \times Y \times 3 \times 3$	$\mathrm{T m^{-1} \mu_0^{-1}}$	yes	Gradient strength of the selection field in x, y, and z directions
`numAverages`	`Int64`	1	1	no	Number of block averages per drive-field period.
`numFrames`	`Int64`	1	1	no	Number of available measurement frames N
`numPeriodsPerFrame`	`Int64`	1	1	no	Number of drive-field periods within a frame denoted by J
`offsetField`	`Float64`	$J \times Y \times 3$	$\mathrm{T \mu_0^{-1}}$	yes	Offset field applied
`startTime`	`String`	1	yyyy-mm-ddThh:mm:ss.ms	no	UTC start time of MPI measurement

2.5.1 Drive Field Parameters (group: `/acquisition/drivefield/`, non-optional)

Remarks: The drive field subgroup describes the excitation details of the imaging protocol. On the lowest level, each MPI scanner contains D channels for excitation. Since most drive-field parameters may change from period to period, they have a leading dimension J.

These excitation signals are usually sinusoidal and can be described by D amplitudes (drive field strengths), D phases, a base frequency, and D dividers. In a more general setting, the generated drive field of channel d can be described by

$$H_d(t) = \sum_{l=1}^{F} A_l \Lambda_l (2\pi f_l t + \varphi_l)$$

where F is the number of frequencies on the channel, A_l is the drive-field strength, ϕ_l is the phase, f_l is the frequency (`baseFrequency`/`divider`$_l$), and Λ_l is the waveform. The waveform is specified by a dedicated parameter `waveform`. It can be set to *sine*, *triangle*, or *custom*.

Parameter	Type	Dim	Unit/Format	Optional	Description
`baseFrequency`	`Float64`	1	Hz	no	Base frequency to derive drive field frequencies
`cycle`	`Float64`	1	s	no	Trajectory cycle is determined by lcm(`divider`)/`baseFrequency`. It will not change when averaging was applied. The duration for measuring the V data points (i.e. the drive-field period) is given by the product of `period` and `numAverages`
`divider`	`Int64`	$D \times F$	1	no	Divider of the `baseFrequency` to determine the drive field frequencies
`numChannels`	`Int64`	1	1	no	Number of drive field channels, denoted by D
`phase`	`Float64`	$J \times D \times F$	rad, $[-\pi, \pi)$	no	Applied drive field phase φ
`strength`	`Float64`	$J \times D \times F$	Tμ_0^{-1}	no	Applied drive field strength
`waveform`	`String`	$D \times F$	1	no	Waveform type: *sine*, *triangle* or *custom*

2.5.2 Receiver (group: `/acquisition/receiver/`, non-optional)

Remarks: The receiver subgroup describes the details on the MPI receiver. For a multi-patch sequence, it is assumed, that the signal acquisition only takes place during particle excitation. During each drive-field cycle, C receive channels record some quantity related to the magnetization dynamic. In most cases these, will be continuous voltage signals induced into the C receive coils, which are proportional to the change of the particle magnetization. This analog signal will usually be converted by some sort of analog to digital converter (ADC) to a discrete series of integer numbers r_{ci} for each channel c. To map the these values to the MPI measurement signal u_{ci}^{ADC}, one has to scale the numbers r_{ci} and add an offset factor

$$u_{ci}^{ADC} = a_c r_{ci} + b_c.$$

Here, a_c and b_c are the scaling and offset factors corresponding to channel c, which can be stored in `dataConversionFactor`. In case the conversion was already performed the `dataConversionFactor` can be ommited. The bandwidth of the receiver is stored in `bandwidth` and is defined as the limit of the first

Nyquist zone. With a sampling rate f_s this would be $f_s/2$. If a decimation d is applied after sampling, the bandwidth is equal to $f_s/d/2$.

The MPI measurement signal is acquired at V equidistant time points. For inductive measurement systems the signal is usually not measured directly at the receive coils but amplified and filtered first, which may damp and distort the signal. Therefore, a transfer function can be stored in the parameter `transferFunction`, which relates the Fourier domain voltage induced at the receive coil $\hat{u}_k^{\text{coil}}$ and the Fourier domain voltage $\hat{u}_k^{\text{ADC}}$ measured at the ADC by

$$\hat{u}_k^{\text{ADC}} = a_k \hat{u}_k^{\text{coil}}, \quad k = 1, \dots, K.$$

Here, a_k are the unitless parameters stored in `transferFunction` for each receive channel individually.

For MPS systems one can additionally store a parameter that maps the induced voltage to the mean magnetic moment of a magnetic nanoparticle located at the center of the scanner. More precisely, in each receive coil a projection of the mean magnetic moment onto the coil sensitivity is measured. The relation of this projection and the voltage at the receive coil in frequency space representation is given by

$$\hat{u}_k^{\text{coil}} = 2\pi \mathrm{i} k \beta \hat{m}_k^{\text{proj}}, \quad k = 1, \dots, K.$$

where $\hat{m}_k^{\text{proj}}$ is the orthogonal projection of the magnetic moment onto the coil sensitivity at the scanner center and β is the channel dependent conversion factor that is stored in the parameter `inductionFactor`.

Parameter	Type	Dim	Unit/Format	Optional	Description
`bandwidth`	Float64	1	Hz	no	Bandwidth of the receiver unit
`dataConversionFactor`	Float64	$C \times 2$	`unit`	yes	Dimension less scaling factor and offset (a_c, b_c) to convert raw data into a physical quantity with corresponding unit of measurement `unit`
`inductionFactor`	Float64	C	`unit` $\mathrm{A}^{-1}\mathrm{m}^{-2}$	yes	Induction factor mapping the projection of the magnetic moment to the voltage in the receive coil
`numChannels`	Int64	1		no	Number of receive channels C
`numSamplingPoints`	Int64	1		no	Number of sampling points during one period, denoted by V
`transferFunction`	Complex128	$C \times K$		yes	Transfer function of the receive channels in Fourier domain with corresponding unit of measurement `unit`
`unit`	String	1		no	SI unit of the measured quantity, usually Voltage V

2.6 Measurement (group: `/measurement/`, optional)

Remarks: MPI data is usually acquired by a series of foreground measurements and optional background measurements. Here, we refer to background measurements as MPI data captured, when any signal generating material, e.g. a phantom or a delta sample is removed from the scanner bore. Initially, all data is available in time domain, where the data of a single frame consists of the signal recorded for all periods in each receive channel, i.e. $J \times C \times V$ data points per set with the temporal index being the fastest to access. If several measurements are acquired (indicated by `numFrames`), the frame dimension is the slowest to access. Along this dimension, the frames are ordered with respect to the time at which they were acquired starting with the measurement acquired first and stopping with the measurement acquired last. We refer to this data as raw measurement data. In Fourier representation, each frame would be stored by $J \times C \times K$ complex data points and $K = V/2 + 1$.

Often it is not convenient to store the raw data but to perform certain processing steps and store the processed data. These steps may lead to a reduction in the number of sampling points from V to W or a corresponding reduction of frequency components K depending on the final representation in which the raw/processed data is stored. The most common processing steps are:

1. Spectral leakage correction, which may be applied to ensure that each individual frame is periodic.
2. Background correction, where the background signal in subtracted.
3. Fourier transformation bringing the data from time into the Fourier representation and storing them in Fourier representation.
4. Transfer function correction to obtain the magnetic moment or induced voltage that has been measured.
5. Frequency selection to reduce the number of frequency components, e.g. bandwidth reduction or selection of high-signal frequency components.
6. Dimension permutation, which is usually applied to Fourier transformed data exchanging the storing order of the data for fast access to the frames.
7. Frame permutation to reorder the frames within the data set.
8. Compression of the O foreground frames by applying a sparsity transformation and storing only significant coefficients.

For each of the steps above there is a corresponding flag within this group indicating if the corresponding processing step has been carried out.

During processing one might want to keep track which of the final N frames belong to background measurements and which do not. To this end, the binary mask `isBackgroundFrame` should be used. If frequency selection has been performed, `frequencySelection` stores the K frequency components (subset) selected from the set of acquired frequency components. If performed, a frame permutation can be described by a bijective mapping $\sigma : \{1, 2, \dots, N\} \to \{1, 2, \dots, N\}$ from the set of frame indices to itself. If such a permutation is performed, σ is stored in the one-line notation as $\sigma(1)$, $\sigma(2)$, $\dots$, $\sigma(N)$ in `framePermutation`.

The motivation for reducing the number of frequency components is the reduction of the file size. Another even more effective method is to compress the data by applying a (linear) sparsity transformation to the data along the frame dimension and storing only the most significant of the resulting coefficients. This method was initially proposed in [8] where a global compression of the system matrix was used. This specification follows the concept introduced in [9], where a fixed number of coefficients is stored for each frequency component. The compression option, indicated by the flag `isSparsityTransformed`, is only available if the processing stages indicated by `isFastFrameAxis`, `isFourierTransformed` have been applied. Furthermore, the frames have to be ordered in such a way that the leading O frames are the foreground frames and the last E frames are the background frames. The order of the frames can be verified by checking the boolean vector `isBackgroundFrame`. The sparsity transformation is only applied to the O foreground frames.

Let $\hat{u}_{j,c,k,o}$ be the measured foreground data in frequency domain with $j = 1, \dots, J$, $c = 1, \dots, C$, $k = 1, \dots, K$, and $o = 1, \dots, O$. The processing indicated by `isSparsityTransformed` is mathematically described by

$$\tilde{u}_{j,c,k,n} = \sum_{o=1}^{O} \Gamma_{n,o} \hat{u}_{j,c,k,o}, \quad \text{for} \quad n = 1, \dots, O$$

where $\mathbf{\Gamma} = (\Gamma_{n,o})_{n,o=1,\dots,O}$ is the transformation matrix. Instead of storing $\tilde{u}_{j,c,k,n}$ completely, the field `data` contains a subset $\tilde{u}_{j,c,k,\beta_{j,c,k,b}}$ where $\beta_{j,c,k,b}$ with $b = 1, \dots, B$ contains the subsampling indices, which are stored in the field `subsamplingIndices`. In order to recover the data $\hat{v}_{j,c,k,o} \approx \hat{u}_{j,c,k,o}$, one first needs to restore $\tilde{v}_{j,c,k,n}$ by

$$\tilde{v}_{j,c,k,n} = \begin{cases} \tilde{u}_{j,c,k,n}, & \text{if } n = \beta_{j,c,k,b} \\ 0, & \text{otherwise.} \end{cases}$$

Then, the inverse transformation $\mathbf{\Gamma}^{-1}$ needs to be applied. In case of unitary transformations, the inversion can be done by applying the adjoint transformation $\mathbf{\Gamma}^{\mathsf{H}}$, i.e.

$$\hat{v}_{j,c,k,o} = \sum_{n=1}^{O} \overline{\Gamma}_{n,o} \tilde{v}_{j,c,k,n}, \quad \text{for} \quad o = 1, \dots, O$$

We note that $\hat{v}_{j,c,k,o}$ is only an approximation of $\hat{u}_{j,c,k,o}$ and that the accuracy of this approximation depends on the compression rate that is given by $\frac{B}{O}$.

The name of the applied transformation $\mathbf{\Gamma}$ is stored in the field `sparsityTransformation`. Currently, the following transformations are sup-

ported: DCT-I, DCT-II, DCT-III, DCT-IV. In all cases we consider the orthogonal / unitary variant of the transformation. For details on the precise definition of the transformations, we refer to the documentation of the FFTW library [10]. Additional transformations are not standardized and must be integrated into this specification before use. In case that the O foreground frames lay on a 2D or 3D grid we consider the multidimensional versions of the transformations. The dimensionality of the transformation can be derived from the number of non-singleton dimensions in `/calibration/size`.

Parameter	Type	Dim	Optional	Description
`data`	`Number`	$N \times J \times C \times K$ or $J \times C \times K \times N$ or $N \times J \times C \times W$ or $J \times C \times W \times N$ or $J \times C \times K \times (B+E)$	no	Measured data at a specific processing stage
`framePermutation`	`Int64`	N	`isFramePermutation`	Indices of original frame order
`frequencySelection`	`Int64`	K	`isFrequencySelection`	Indices of selected frequency components
`isBackgroundCorrected`	`Int8`	1	no	Flag, if the background has been subtracted
`isBackgroundFrame`	`Int8`	N	no	Mask indicating for each of the N frames if it is a background measurement (true) or not
`isFastFrameAxis`	`Int8`	1	no	Flag, if the frame dimension N has been moved to the last dimension
`isFourierTransformed`	`Int8`	1	no	Flag, if the data is stored in frequency space
`isFramePermutation`	`Int8`	1	no	Flag, if the order of frames has been changed, see `framePermutation`
`isFrequencySelection`	`Int8`	1	no	Flag, if only a subset of frequencies has been selected and stored, see `frequencySelection`
`isSparsityTransformed`	`Int8`	1	no	Flag, if the foreground frames are compressed along the frame dimension
`isSpectralLeakageCorrected`	`Int8`	1	no	Flag, if spectral leakage correction has been applied
`isTransferFunctionCorrected`	`Int8`	1	no	Flag, if the data has been corrected by the `transferFunction`
`sparsityTransformation`	`String`	1	`isSparsityTransformed`	Name of the applied sparsity transformation
`subsamplingIndices`	`Integer`	$J \times C \times K \times B$	`isSparsityTransformed`	Subsampling indices $\beta_{j,c,k,b}$

2.7 Calibration (group: `/calibration/`, optional)

Remarks: The calibration group describes a calibration measurement (system matrix), although it does not hold the data itself. Each of the calibration measurements is taken with a calibration sample (delta sample) at a grid centered position inside the FOV of the device. If available, background measurements are taken with the delta sample outside of the FOV of the scanner. Usually, not the raw measurements are stored but processed data, where at least averaging, Fourier transformation, frame permutation and transposition has been performed yielding a total of N processed frames. Of these frames O frames correspond to the calibration scans at the O spatial positions, whereas the other frames correspond to background measurements taken throughout the calibration process. All processing steps are documented in the `/measurement` group.

If the measurements were taken on a regular grid of size $N_x \times N_y \times N_z$, the permutation is usually done such that measurements are ordered with respect to their x position first, second with respect to their y position, and last with respect to their z position. Background measurements are collected at the end in `/measurement/data`, which in combination with reordering of the measurements allows a fast access to the system matrix. If a different storage order is used this can be documented using the optional parameter `order`. For non-regular sampling points, there is the possibility to explicitly store all O positions. If the calibration measurement is performed in a MPS system, the spatial positions are usually emulated by applying offset fields, which can be stored in `offsetFields`.

Parameter	Type	Dim	Unit/Format	Optional	Description
`deltaSampleSize`	`Float64`	3	m	yes	Size of the delta sample used for calibration scan
`fieldOfView`	`Float64`	3	m	yes	Field of view of the system matrix
`fieldOfViewCenter`	`Float64`	3	m	yes	Center of the system matrix (relative to origin/center)
`method`	`String`	1		no	Method used to obtain calibration data. Can for instance be robot, hybrid, or simulation
`offsetFields`	`Float64`	$O \times 3$	$\mathrm{T}\mu_0^{-1}$	yes	Applied offset field strength to emulate a spatial position (x, y, z)
`order`	`String`	1		yes	Ordering of the dimensions, default is xyz
`positions`	`Float64`	$O \times 3$	m	yes	Position of each of the grid points, stored as (x, y, z) triples
`size`	`Int64`	3		yes	Number of voxels in each dimension, inner product is O
`snr`	`Float64`	$J \times C \times K$		yes	Signal-to-noise estimate for recorded frequency components

2.8 Reconstruction Results (group: `/reconstruction/`, optional)

Reconstruction results are stored using the parameter `data` inside this group. `data` contains a $Q \times P \times S$ array, where Q denotes the number of reconstructed frames within the data set, P denotes the number of voxels and S the number of multispectral channels. If no multispectral reconstruction is performed, then one may set $S = 1$. Depending on the reconstruction the grid of the reconstruction data can be different from the system matrix grid. Hence, grid parameters are mirrored in the `/reconstruction` group.

For analysis of the MPI tomograms, it is often required to know which parts of the reconstructed tomogram have been covered by the trajectory of the field free region. In MPI, one refers to the non-covered region as overscan region. Therefore, the optional binary field `isOverscanRegion` stores for each voxel if it is part of the overscan region. If no voxel lies within the overscan region, `isOverscanRegion` may be omitted.

Parameter	Type	Dim	Unit/Format	Optional	Description
`data`	`Number`	$Q \times P \times S$		no	Reconstructed data
`fieldOfView`	`Float64`	3	m	yes	Field of view of reconstructed data
`fieldOfViewCenter`	`Float64`	3	m	yes	Center of the reconstructed data (relative to scanner origin/center)
`isOverscanRegion`	`Int8`	P		yes	Mask indicating for each of the P voxels if it is part of the overscan region (true) or not
`order`	`String`	1		yes	Ordering of the dimensions, default is xyz
`positions`	`Float64`	$P \times 3$	m	yes	Position of each of the grid points, stored as (x, y, z) tripels
`size`	`Int64`	3		yes	Number of voxels in each dimension, inner product is P

3 Changelog

3.1 v2.1.0

- Added the possibility to store compressed calibration data using a sparsity transformation. To this end, the `data` field in the `measurement` group has been extended and the fields `isSparsityTransformed`, `sparsityTransformation` and `subsamplingIndices` have been introduced.

3.2 v2.0.1

- Added optional `time` field to `study` group to be able to document the creation time of the study.
- Lexicographical reordering of all fields within each group to improve readability.

3.3 v2.0.0

- Version 2 of the MDF is a major update breaking backwards compatibility with v1.x. The major update was necessary due to several shortcomings in the v1.x.
- The naming of parameters was made more consistent. Furthermore, some parameters moved from one group into another.
- Defined a complex datatype using a HDF5 compound type.
- In v1.x it was not possible to store background data. This functionality has been added in v2.
- We simplified the measurement group and made it much more expressive. In v1.x it was not entirely clear, which processing steps have been applied to the measurement data in the stored dataset. The `measurement` group now contains several flags that precisely document the state of the stored data. Using this it is now possible to also store calibration data in the `measurement` group. The calibration group in turn only stores metadata about a calibration experiment while the actual data is store in the measurement group.
- Updated Affiliations in the MDF specification.
- Improved the general descriptions of fields and groups.
- In v1.x the MDF allowed many fields to have varying dimensions depending on the context. As of version 2.0.0 only one field offers this freedom. This change should make implementations handling MDF files less complex.
- `Number` has been introduced as a generic type.
- Added a table listing all variable names used in the descriptions of parameters.
- Added a section describing the possibility to add user defined parameters to MDF files.
- Added a description for optional and non-optional groups and conditional, optional, and non-optional datasets.
- Added a short section on the code examples on the Github repository.
- Support for triangle wave forms has been added.
- Support for multiple excitation frequencies on a drive-field channel has been added.
- Added the dimension A to all fields of the `tracer` group to be able to describe settings where multiple tracers are used or tracers are administered multiple times.
- Added the possibility to store the tracer concentration also for non iron based tracer materials by adding the `/tracer/solute` field and redefining the field `tracer/concentration`.
- Improved documentation for the storage of ADC transfer functions. It is also possible now to store the measurement data as integer data and use a `dataConversionFactor` to describe the mapping to a physical representation (e.g. Volt)
- Added support for receive coil to ADC transfer functions.
- Added support for mean magnetic moment to receive coil voltage transfer functions.

- Split the `/study` group into the `/experiment` group and the study group. This allows to provide more fine grained information on study and experiment.
- Added possibility to mark the overscan region.
- Added new section changelog to the MDF documentation to record the development of the MDF.
- Updated `README.md` and `MDF.bib` in the github repository.
- Updated code examples in the github repository.
- Added a section on the MDF reference implementation MPIFiles.jl. Since sanity checks will be covered by this package, the description on sanity checks has been removed.

3.4 v1.0.5

- Added the possibility to store different channels of reconstructed data.
- Added support for receive channels with different characteristics (e.g. bandwidth).
- Made dataset `/acquisition/receiver/frequencies` optional.
- Extended the description on the data types, which are used to store data.
- Added references for Julia and HDF5 to the specifications.

3.5 v1.0.4

- Clarify that HDF5 datasets are used to store MPI parameters.

3.6 v1.0.3

- Updated Affiliations in the MDF specification.
- Included data download into the Python and Matlab example code.
- Changes in the Python and Matlab example code to be better comparable to the Julia example code.

3.7 v1.0.2

- Added reference to arXiv paper and bibtex file for reference.

3.8 v1.0.1

- A sanity check within the Julia code shipped alongside the specifications.
- An update to the specification documenting the availability of a sanity check.
- Updated MDF files on https://www.tuhh.de/ibi/research/mpi-data-format.html.
- Updated documentation to the Julia, Matlab and Python reconstruction scripts.
- Improved Julia reconstruction script, automatically downloading the required MDF files.

References

[1] The HDF Group. Hierarchical Data Format, version 5, 1997-2016. http://www.hdfgroup.org/HDF5/.

[2] P. J. Leach, M. Mealling, and R. Salz. A universally unique identifier (UUID) URN namespace. 2005.

[3] B. Gleich and J. Weizenecker. Tomographic imaging using the nonlinear response of magnetic particles. *Nature*, 435(7046):1214–1217, 2005.

[4] T. Knopp, M. Möddel, F. Griese, F. Werner, P. Szwargulski, N. Gdaniec, and M. Boberg. MPIFiles.jl: A julia package for magnetic particle imaging files. *Journal of Open Source Software*, 4(38):1331, 2019.

[5] J. Bezanson, S. Karpinski, V. B. Shah, and A. Edelman. Julia: A fast dynamic language for technical computing. *CoRR*, abs/1209.5145, 2012.

[6] J. Bezanson, J. Chen, S. Karpinski, V. B. Shah, and A. Edelman. Array operators using multiple dispatch: a design methodology for array implementations in dynamic languages. *CoRR*, abs/1407.3845, 2014.

[7] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *CoRR*, abs/1411.1607, 2014.

[8] J. Lampe, C. Bassoy, J. Rahmer, J. Weizenecker, H. Voss, B. Gleich, and J. Borgert. Fast reconstruction in magnetic particle imaging. *Physics in Medicine and Biology*, 57(4):1113, 2012.

[9] T. Knopp and A. Weber. Local system matrix compression for efficient reconstruction in magnetic particle imaging. *Advances in Mathematical Physics*, 2015, 2015.

[10] M. Frigo and S. G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".